HACKING LAB
特典
ハッキング・ラボに
役立つテクニック＋α

1 Hyper-VとVirtualBoxの共存

Hyper-Vとは、Windowsに備わっているハイパーバイザー型の仮想化ソフトです。

VirtualBoxでは仮想マシンをvhdファイルで管理します。また、Hyper-Vでもvhdファイルを扱えます。つまり、VirtualBoxの資源（仮想マシン）をHyper-Vでそのまま利用できます。しかし、Hyper-VとVirtualBoxは共存できません。Hyper-Vを使うために機能（後述するWindowsのHyper-V機能）を有効にすると、VirtualBoxを起動するときにエラーになるからです（*1）。

そのため、本書では、Hyper-Vを利用できるかを確認する方法、Hyper-V機能を有効にする方法までを説明します。完全にHyper-Vに移行するかどうかは読者の判断にまかせます。

Hyper-Vを利用できる条件

Hyper-Vを使うためには、次の条件を満たす必要があります。

OS

- Windows 10の64bit版である。
- エディションがPro以上（Pro、Enterprise、Education）である。

ハードウェア

- CPUが仮想化を実現する機能をサポートしている。
  第2レベルのアドレス変換（SLAT）の64ビットCPUである。
  VMモニターモード拡張機能（Intel CPUのVT-c）をサポートしている。
- 4Gバイト以上のメモリーを持つ。
- BIOS/UEFIが仮想化テクノロジーをサポートしている。
  Hyper-V機能を利用するための命令をハードウェア側が認識できる。
- データ実行防止機能を使用できる。

*1："VT-x is not available." というエラーが表示され、VirtualBoxが起動できなくなります。

》》Hyper-Vに対応しているかを確認する

ハードウェアがHyper-Vに対応しているかどうかは、次の方法で確認できます。

●方法1

Windows 10であれば、タスクマネージャーのパフォーマンスタブに仮想化（Virtualization）という項目があります。ここが「有効」（Enabled）なら、CPUが仮想化をサポートし、BIOSでそれが有効になっていることを意味します（図1）。もしこの項目がなかったら、CPUが仮想化をサポートしていません。
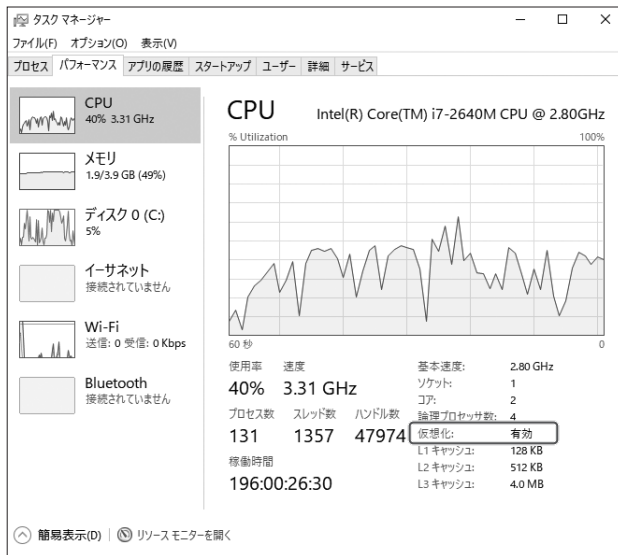
図1　タスクマネージャーで確認する

●方法2

Windows 10の検索ボックスに「msinfo32」と入力して、システム情報画面を表示します。Hyper-V関連の4項目がすべて「はい」であることを確認します（図2）。
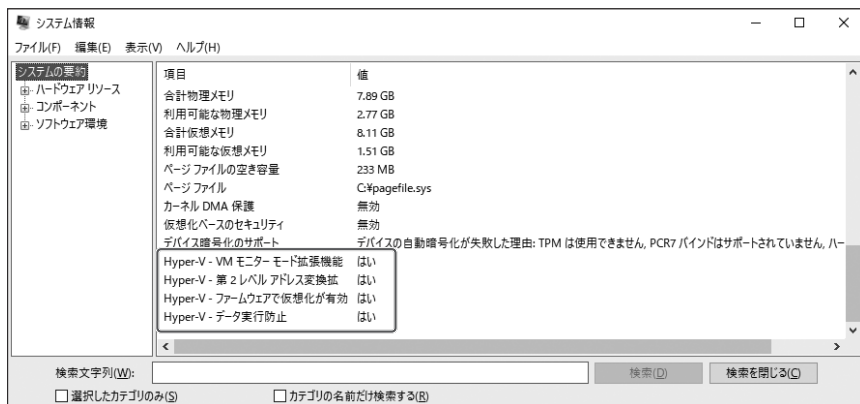
図2 「msinfo32」で確認する

また、BIOSの画面では、図3のように表示されます。ただし、メーカーによって表示内容は異なります。
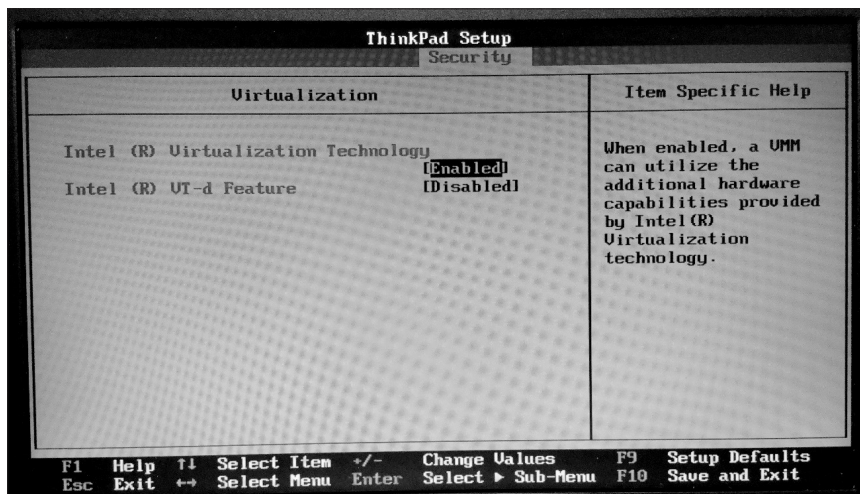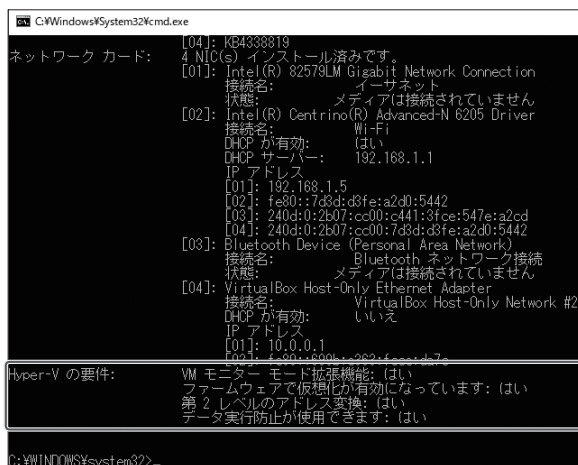
図3 BIOSの仮想化テクノロジーのサポート

●方法3

コマンドプロンプトでsysteminfoコマンドを実行します。システム情報の内容が表示されます（方法2で表示されるシステム情報画面の内容が出力される）。出力内容が多すぎて読み取れないのであれば、moreコマンドも併用します。

```
C:¥Users¥ipusiron>systeminfo | more
```

最後の方に「Hyper-Vの要件」（Hyper-V Requirements）という項目があります（図4）。

図4　systeminfoで確認する

》Hyper-V機能の有効化

デフォルトではHyper-V機能が無効になっているので、次の手順で有効にします。

①Hyper-V機能を有効化する

コントロールパネルにて「プログラムと機能」を選びます。「Windowsの機能の有効化または無効化」を選び、Hyper-Vにチェックを入れます（図5）。
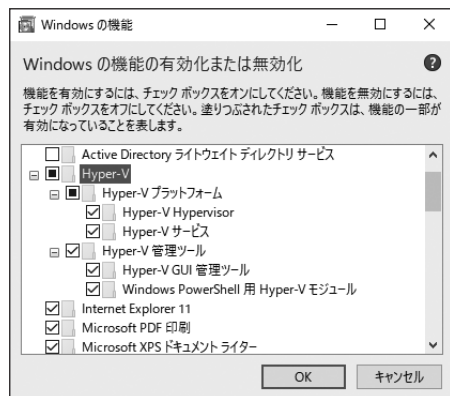
図５　Hyper-V機能の有効化

②再起動する

指示にしたがい、再起動します。これでHyper-Vを起動できます。

もし再起動しないでもHyper-V機能を有効にできるのであれば、若干の手間はかかりますがHyper-VとVirtualBoxを共存できたでしょう。しかしながら、再起動は必須のため、効率の面から共存できないといえます。

2 VagrantによるKaliの仮想マシンの構築

Vagrantとは何か

Vagrantとは、仮想マシンを簡単に構築するためのソフトウェアです。VagrantはVirtualBoxのフロントに位置し、代理でVirtualBoxを操作してくれるものといえます。Boxと呼ばれるテンプレートのようなものを用いることで、コマンド1つで仮想環境を構築できます。

Vagrant by HashiCorp
https://www.vagrantup.com/

Vagrantの特徴

Vagrantは便利なツールですが、メリットだけでなく、デメリットもあります。

●メリット

環境構築を自動化できる

環境構築を自動化できます。例えば、開発環境を構築する際に、Vagrantであればコマンド1発で開発用の仮想環境を構築できます。そして、Chefやpuppetなどの構成管理ツールと連携できます。さらに、仮想環境であるため、チーム内で同一の環境を構築できます。

設定を再利用しやすい

仮想マシンの設定をファイルに書き込むため、設定を再利用しやすいといえます。つまり、仮想マシンを破棄してもすぐに再構築できます。

インターフェースが変わらない（CUI）

Virtual BoxはGUIなのである程度直感的に使用できます。しかし、バージョンアップするとインターフェースが変わることがあります。個人レベルの用途であれば、大幅にインターフェースが変わらない限り問題はありません。しかし、開発や運用の現場では、環境構築の手順書が用意されることがあります。インター

フェースが変わってしまうと、手順書の画面説明を更新するという手間が生じてしまいます。

バージョン管理ができる

開発の現場で仮想マシンにその場しのぎの更新をしたり、管理する人が入れ替わったりするうちに、構成がブラックボックス化してしまいます。仮想マシンを更新しても、その更新内容を明文化しておかないとわからなくなってしまいます。結局、開発環境と運用環境にギャップが出てしまい、運用テストのときに深刻なバグが発見されてしまうかもしれません。

一方、Vagrantであれば、Vagrantfileファイルをバージョン管理できます。つまり、誰が更新して、どんな差分が生じたのかが明白であるため、構成がブラックボックス化しにくいといえます。さらに、Vagrantfileファイルのみであればファイル容量は小さいので、仮想マシンを丸ごと共有するより効率がよいといえます。

●デメリット

VirtualBoxのバージョンをサポートしている必要がある

VagrantとVirtualBoxは密接に関係するため、VirtualBoxのバージョンがVagrantでサポートされていなければなりません。特に、VirtualBoxをアップデートする際に気を付ける必要があります。

VagrantとVirtualBoxのバージョンの相性問題については、検索して動作がうまくいっている組み合わせを参考にするとよいでしょう。過去のバージョンのVagrantは次のURLからダウンロードできます。

Vagrant Versions | HashiCorp Releases
https://releases.hashicorp.com/vagrant/

Vagrantのインストール

ホストOSのWindowsにVagrantをインストールします。

①インストーラーをダウンロードする

Vagrantのインストーラーをダウンロードします。最新のバージョンであればHashiCorpのVagrantのページ（https://www.vagrantup.com）からダウンロードします。過去のバージョンのものであれば、https://releases.hashicorp.com/vagrant/からダウンロードできます。Windows向けのインストーラーは、拡張子が".msi"になっています。

②インストールする

ダウンロードしたインストーラーを実行します。インストール後、再起動します。

③バージョンが表示されることを確認する

コマンドプロンプトで次のように入力して、バージョンが表示されることを確認します。

```
>vagrant --version
```

初めてのVagrant

コマンドプロンプトやPowerShellでVagrantを操作できますが、本書ではgit bashを用います（*2）。

①プロジェクトのフォルダーを作成する

git bashを起動して、プロジェクトのフォルダーを作成します。プロジェクトの保存フォルダーのパスは、"C:¥vagrant¥first_vagrant"にします。

```
ipusiron@Garoa MINGW64 ~
$ pwd
/c/Users/ipusiron
```

*2：Gitのインストール時にgit bashもインストールしているはずです。

```
ipusiron@Garoa MINGW64 ~
$ cd /C

ipusiron@Garoa MINGW64 /C
$ mkdir vagrant

ipusiron@Garoa MINGW64 /C
$ cd vagrant/

ipusiron@Garoa MINGW64 /C/vagrant
$ mkdir first_vagrant

ipusiron@Garoa MINGW64 /C/vagrant
$ ll
total 0
drwxr-xr-x 1 ipusiron 197121 0 7月   6 22:30 first_vagrant/

ipusiron@Garoa MINGW64 /C/vagrant
$ cd first_vagrant/
```

②プロジェクトを作成する

このステップではプロジェクトを作ります。カレントディレクトリをプロジェクトのフォルダーに移動してから、ベースとなる仮想マシンイメージを指定します。このイメージをVagrantの世界ではBoxといいます。この時点でVagrantfileが生成されます。

```
ipusiron@Garoa MINGW64 /C/vagrant/first_vagrant
$ vagrant init centos/7   ←CentOS7の仮想マシンイメージを指定する。
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

③Boxをダウンロードする

vagrant upコマンドで仮想マシンを起動します。

```
ipusiron@Garoa MINGW64 /C/vagrant/first_vagrant
$ ll
total 4                                              Vagrantfileが生成された。
-rw-r--r-- 1 ipusiron 197121 3429 7月   6 22:48 Vagrantfile ←
ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Box 'centos/7' could not be found. Attempting to ↵
find and install...
（略）
==> default: Adding box 'centos/7' (v1804.02) for provider: ↵
virtualbox
（略）
==> default: Rsyncing folder: /cygdrive/c/vagrant/↵
first_vagrant/ => /vagrant
```

Boxがまだダウンロードされていないので、vagrant upコマンドの実行後に自動でダウンロードが始まります。ダウンロードが終わるまで少し時間がかかります。

なお、事前にBoxをダウンロードしておく場合は、vagrant box add <Box名>コマンドを用います。addしたBoxはvagrant box listコマンドで確認できます。

もしvagrant upコマンドの実行時に、「404 Not Found」というエラーが出ていれば、ダウンロードに失敗しています。原因は色々考えられます。Vagrantのバージョンが古い、アクセスするURLが変わっているなどです（*3）。rmコマンドでVagrantfileファイルを削除して、問題を改善してから、vagrant initコマンドからやり直してください。

ダウンロードが完了すると、仮想マシンが起動します。すでにダウンロード済

*3：過去にBoxを提供していたVagrant Atlasサービスが、Vagrant Cloudに変わったという事例もあります。筆者の環境の場合、「Windows 7＋Vagrantバージョン1.95」でエラーが出ましたが、バージョン1.98にすることで改善しました。

みであれば、次回からのvagrant upコマンドの実行ですぐに仮想マシンが起動します。このとき、自動でSSHの設定や共有フォルダーのマウント処理が適用されます。

④仮想マシンの状態を確認する

仮想マシンの状態を確認します（*4）。

```
ipusiron@Garoa MINGW64 /C/vagrant/first_vagrant
$ vagrant status
Current machine states:

default                   running (virtualbox)  ← 仮想マシンの状態がわかる。

The VM is running. To stop this VM, you can run `vagrant halt` to
shut it down forcefully, or you can run `vagrant suspend` to simply
suspend the virtual machine. In either case, to restart it again,
simply run `vagrant up`.
```

"running" となっているので、仮想マシンは動作しています。

さらに、ダウンロードしたBoxがVagrantに登録されていることを確認します。

```
ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant box list
centos/7 (virtualbox, 1804.02)
```

この時点でVirtualBoxを起動すると、"first_vargrant_default_XXXX…"（Xは数字）という仮想マシンが追加されています。そして、vagrant upコマンドやvagrant haltコマンドに合わせて、状態が実行中・電源オフに変化します（図6）。

*4：vagrant statusコマンドを実行すると、Vagrantの仮想マシンが列挙され、それぞれの動作状態が表示されます。ディレクトリパスも表示したい場合には、vagrant global-statusコマンドを用います。
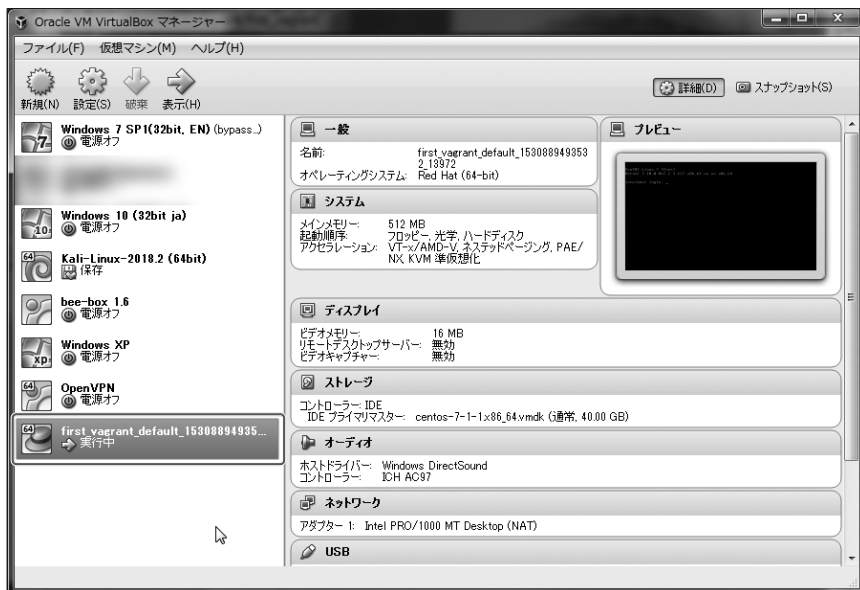
図6　Vagrantで作成した仮想マシン

⑤仮想マシンにアクセスできることを確認する

vagrant sshコマンドで仮想マシンにアクセスできることを確認します。

```
ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant ssh
[vagrant@localhost ~]$  ← SSHでログインした。
[vagrant@localhost ~]$ ping akademeia.info  ←
                                   インターネットにアクセスできることを確認した。
PING akademeia.info (59.106.19.200) 56(84) bytes of data.
64 bytes from www750.sakura.ne.jp (59.106.19.200): icmp_seq=1
ttl=52 time=29.2 ms
64 bytes from www750.sakura.ne.jp (59.106.19.200): icmp_seq=2
ttl=52 time=32.3 ms
--- akademeia.info ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
```

```
rtt min/avg/max/mdev = 29.253/30.813/32.373/1.560 ms
[vagrant@localhost ~]$ whoami
vagrant
[vagrant@localhost ~]$ su  ←rootに切り替えたい。
Password:  ←パスワードとして "vagrant" を入力する(*5)。
[root@localhost vagrant]# whoami
root
[root@localhost vagrant]# exit
exit
[vagrant@localhost ~]$ exit
logout
Connection to 127.0.0.1 closed.

ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ whoami
ipusiron
```

⑥サスペンドする / レジュームする

vagrant suspendコマンドやvagrant resumeコマンドの使い方を解説します。

サスペンド（suspend）した場合、仮想マシンの状態を保持したまま、すなわちメモリーの内容を保持したまま終了するので、容量を消費します。ただし、レジューム（resume）するとすばやく復帰し、状態を復元できます。

```
ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant suspend  ←サスペンドする。
==> default: Saving VM state and suspending execution...

ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant status
Current machine states:
```

*5：suコマンドがうまくいかない場合は、sudo suを実行します。

```
default                          saved (virtualbox)  ← savedに変化した。

To resume this VM, simply run `vagrant up`.

ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant ssh
VM must be running to open SSH connection. Run `vagrant up`
to start the virtual machine. ←
                                          仮想マシンはサスペンド状態なのでSSHで接続できない。
ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant resume  ← レジュームする。
==> default: Resuming suspended VM...
（略）
==> default: flag to force provisioning. Provisioners marked to ↵
run always will still run.

ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant status
Current machine states:

default                          running (virtualbox)  ← runningに戻った。

The VM is running. To stop this VM, you can run `vagrant halt` to
shut it down forcefully, or you can run `vagrant suspend` to ↵
simply
suspend the virtual machine. In either case, to restart it again,
simply run `vagrant up`.
```

⑦仮想マシンを終了する

仮想マシンを終了するには、vagrant haltコマンドを用います。

```
ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant halt
==> default: Attempting graceful shutdown of VM...

ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant status
Current machine states:

default                    poweroff (virtualbox)  ← poweroffになっている。

The VM is powered off. To restart the VM, simply run `vagrant up`
```

仮想マシンのサスペンドや終了は、VirtualBoxの仮想マシンの保存や終了に対応していると考えると、直感的にわかりやすいでしょう。

⑧仮想マシンを破棄する

仮想マシンを破棄するには、プロジェクトフォルダー上でvagrant destroyコマンドを実行するだけです。

```
ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant destroy
    default: Are you sure you want to destroy the 'default' VM? ↵
[y/N] y
==> default: Destroying VM and associated drives...

ipusiron@Garoa MINGW64 /c/vagrant/first_vagrant
$ vagrant status
Current machine states:

default                    not created (virtualbox)

The environment has not yet been created. Run `vagrant up` to
create the environment. If a machine is not created, only the
```

```
default provider will be shown. So if a provider is not listed,
then the machine is not created for that environment.
```

仮想マシンを破棄しても、Vagrantfileファイルは残ります。また、Boxはダウンロード済みなので、vagrant upコマンドですぐに作成できます。ここでは、安心してvagrant destroyコマンドを試してみてください。

このように仮想マシンの作成・破棄を簡単にできることがVagrantの魅力の1つです。コマンドだけで操作が完結するので、自動化しやすいといえます（*6）。

KaliのBoxを自作する

Vagrant Cloud（https://app.vagrantup.com/boxes/search）から様々なBoxを探せます。Kaliを検索してください。しかし、古いバージョンのものは見つかりますが、最新のバージョンのものは見つからないかもしれません。その場合は、自分でBoxを作った方が早いでしょう。ここではKali 2018.2のBoxを作る方法を紹介します。

●Boxを自作する

①仮想マシンのフォルダーパスを確認する

VirtualBoxでKaliを起動します。もし初期状態のBoxを作りたいのであれば、Kaliをインストールした直後に行います。

仮想マシンのフォルダーパスを把握しておきます。仮想マシンの「設定」＞「ストレージ」を選びます。ストレージツリーで「コントローラ：SATA」の下のvmdkファイルを選びます。すると、右側の場所にvmdkファイルのパスが表示されます（図7）。本書の通りに仮想マシンを作成した場合は、"C:¥VM_Guest¥VBox¥Kali-Linux-2018.2-vbox-amd64¥Kali-Linux-2018.2-vbox-amd64-disk001.vmdk" になっているはずです。

*6：VirtualBoxでもVBoxManageコマンドを使えば、CUIで仮想マシンを起動したり、スナップショットを作成できたりします。

図7　vmdkファイルの位置を確認する

②ソフトウェアをアップデートする

Kaliを起動して、ソフトウェアをアップデートしておきます。

```
root@kali:~# apt update
root@kali:~# apt upgrade
```

③vagrantユーザーを作成する

vagrantユーザーを作成します。その後、vagrantユーザーにsudo権限を追加します。

```
root@kali:~# useradd -m vagrant
root@kali:~# passwd vagrant
Enter new UNIX password:       ← ここではパスワードを "vagrant" とした。
Retype new UNIX password:      ← 同じパスワードを入力する。
passwd: password updated successfully  ログインシェルを "/bin/bash" にする。
root@kali:~# usermod -s /bin/bash vagrant  ←
root@kali:~# usermod -a -G sudo vagrant  ← sudoグループに属するようにする。
```

④SSHをセットアップする

SSHをセットアップして、ssh vagrantコマンドでアクセスできるようにします。

```
root@kali:~# su - vagrant
vagrant@kali:~$ id
uid=1001(vagrant) gid=1002(vagrant) groups=1002(vagrant),27(sudo)
vagrant@kali:~$ pwd
/home/vagrant
vagrant@kali:~$ mkdir /home/vagrant/.ssh
vagrant@kali:~$ chmod 700 /home/vagrant/.ssh
```

今回は実験用途であり外部に公開するわけではないので、Vagrantで配布されている公開鍵を用います（*7）。これにより、デフォルト設定のままvagrant sshコマンドでSSHアクセスできます。curlコマンドの-kオプションは、SSL証明書の警告を無視することを意味します。-Lオプションはリダイレクト先に再接続し、-oオプションは出力するファイル名です。

```
vagrant@kali:~$ cd /home/vagrant/.ssh/
vagrant@kali:~/.ssh$ curl -k -L -o authorized_keys 'https://raw.github.com/↵
mitchellh/vagrant/master/keys/vagrant.pub'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100   409  100   409    0     0    330      0  0:00:01  0:00:01 --:--:--   330
vagrant@kali:~/.ssh$ ls
authorized_keys
vagrant@kali:~/.ssh$ chmod 600 authorized_keys
vagrant@kali:~/.ssh$ chown -R vagrant:vagrant /home/vagrant/.ssh
```

*7：https://github.com/hashicorp/vagrant/tree/master/keys
外部に公開するといった用途であれば、各ユーザーの鍵に変更します。

SSHを自動起動するように設定します。

```
vagrant@kali:~$ exit
logout
root@kali:~# systemctl enable ssh.service
Synchronizing state of ssh.service with SysV service script ↵
with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable ssh
Created symlink /etc/systemd/system/sshd.service → /lib/↵
systemd/system/ssh.service.
root@kali:~# systemctl start ssh.service
```

⑤パスワードなしでsudoコマンドを実行できるようにする

vagrant upコマンド実行時にpasswordなしでsudoできるように設定を変えます。

visudoコマンドを使って"/etc/sudoers"ファイルを編集します。このファイルは編集をミスすると大変危険であるため、rootでさえ直接変更できません。visudoコマンドを経由して編集することで、保存時に文法チェックもしてくれ、うっかりしたミスを防いでくれます。

```
root@kali:~# visudo
```

sudoグループのユーザーが、sudoコマンドをパスワードなしで実行できるようにするには、次のように編集します。

編集前

```
%sudo   ALL=(ALL:ALL) ALL
```

編集後

```
%sudo   ALL=(ALL) NOPASSWD:ALL
```

vagrantユーザーになってsudoの実行を確認します。

```
root@kali:~# su - vagrant
vagrant@kali:~$ pwd
/home/vagrant
vagrant@kali:~$ sudo ls /root
（ファイルがずらりと表示されれば成功）
vagrant@kali:~$ exit
logout
root@kali:~#
```

⑥不要なファイルを削除する

不要なファイルを削除します。apt-getコマンドからインストールされたファイルは"/var/cache/apt/archives"にキャッシュされていて、アプリを削除した後もキャッシュは残ります。アプリをインストールしたらキャッシュは不要といえるので、これを削除しましょう。

apt-getにはautocleanとcleanというサブコマンドが用意されています。apt-get autocleanコマンドでは、この "/var/cache/apt/archives" にキャッシュされていて、システムにはインストールされていないdebファイルを削除します。一方、apt-get cleanコマンドでは、"/var/cache/apt/archives" にキャッシュされているすべてのパッケージを削除します。

よって、apt-get cleanコマンドを実行すればよいことがわかります。このコマンドは、apt-get autocleanコマンドの削除対象を含んでいるためです。

```
root@kali:~# df -hT  ← キャッシュの削除前に空き容量を確認する。
Filesystem     Type      Size  Used Avail Use% Mounted on
udev           devtmpfs  981M     0  981M   0% /dev
tmpfs          tmpfs     200M   19M  181M  10% /run
/dev/sda1      ext4       77G   21G   53G  28% /
tmpfs          tmpfs     999M     0  999M   0% /dev/shm
tmpfs          tmpfs     5.0M     0  5.0M   0% /run/lock
tmpfs          tmpfs     999M     0  999M   0% /sys/fs/cgroup
```

```
tmpfs           tmpfs     200M   16K  200M   1% /run/user/131
tmpfs           tmpfs     200M   40K  200M   1% /run/user/0
root@kali:~# # apt-get clean  ← キャッシュを削除する。
root@kali:~# df -hT  ← キャッシュの削除後に空き容量を確認する。
Filesystem      Type      Size  Used Avail Use% Mounted on
udev            devtmpfs  981M     0  981M   0% /dev
tmpfs           tmpfs     200M   19M  181M  10% /run
/dev/sda1       ext4       77G   17G   57G  23% /  ← 空き容量が4Gバイトほど増えた。
tmpfs           tmpfs     999M     0  999M   0% /dev/shm
tmpfs           tmpfs     5.0M     0  5.0M   0% /run/lock
tmpfs           tmpfs     999M     0  999M   0% /sys/fs/cgroup
tmpfs           tmpfs     200M   16K  200M   1% /run/user/131
tmpfs           tmpfs     200M   40K  200M   1% /run/user/0
```

⑦ネットワークを設定する

ネットワークを設定します。Vagrantfileファイルでは1つの仮想LANアダプターがNATに割り当てられています。そこで、Kali側も対応するように設定します。eth0は動的にIPアドレスを取得するようにします。

"/etc/network/interfaces"ファイル

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp
```

⑧仮想マシンを終了する

以上で基本的な設定は完了したので、仮想マシンを終了します。

```
root@kali:~# shutdown -h now
```

⑨Boxを作成する

以降はgit bash上の入力です。まず、任意のワークフォルダーに移動します。

```
ipusiron@Garoa MINGW64 ~
$ cd /c/work
```

Boxを作るコマンドの書式は次の通りです（*8）。

```
$ vagrant package --base <作成対象の仮想マシン名> --output ↵
<出力するBoxファイル名>
```

今回の仮想マシン名は "Kali-Linux-2018.2-vbox-amd64" であり、出力するBoxのファイル名は "Kali-Linux-2018.2-amd64.box" とします。作成にはかなり時間がかかります（数十分）。

```
ipusiron@Garoa MINGW64 /c/work
$ vagrant package --base Kali-Linux-2018.2-vbox-amd64 --output ↵
Kali-Linux-2018.2-amd64.box
==> Kali-Linux-2018.2-vbox-amd64: Exporting VM...
==> Kali-Linux-2018.2-vbox-amd64: Compressing package to: C:/↵
work/Kali-Linux-2018.2-amd64.box
```

生成されたBoxファイルは約13Gバイトになりました。

⑩Boxを適切な場所に移動する

生成したBoxは適切な場所に移動します。ところで、vagrantのデフォルトのBoxの配置場所は "C:¥Users¥<ユーザー名>¥.vagrant.d¥boxes" 内になっていま

*8：ファイル名に空白を含む場合は、シングルクォート（「'」）で囲みます。

す。ここに "Kali-Linux-2018.2" というフォルダーを作り、この中に生成したKaliのBoxを移動します。

⑪Boxを登録する

vagrant box listコマンドの出力にKaliのBoxが表示されるように、Boxを登録します。このとき、vagrant box addコマンドを用います。--nameオプションでは仮想マシンの登録名を指定します。

```
ipusiron@Garoa MINGW64 ~
$ cd /c/Users/ipusiron/.vagrant.d/boxes/Kali-Linux-2018.2

ipusiron@Garoa MINGW64 ~/.vagrant.d/boxes/Kali-Linux-2018.2
$ vagrant box list
centos/7 (virtualbox, 1804.02)  ←1つだけある。

ipusiron@Garoa MINGW64 ~/.vagrant.d/boxes/Kali-Linux-2018.2
$ vagrant box add --name Kali-Linux-2018.2 Kali-Linux-2018.2-↵
amd64.box  ←Boxを登録する。
==> box: Box file was not detected as metadata. Adding it ↵
directly...
==> box: Adding box 'Kali-Linux-2018.2' (v0) for provider:
    box: Unpacking necessary files from: file://C:/Users/↵
ipusiron/.vagrant.d/boxes/Kali-Linux-2018.2/Kali-Linux-2018.2-↵
amd64.box
    box:
==> box: Successfully added box 'Kali-Linux-2018.2' (v0) for ↵
'virtualbox'!

ipusiron@Garoa MINGW64 ~/.vagrant.d/boxes/Kali-Linux-2018.2
$ vagrant box list
Kali-Linux-2018.2 (virtualbox, 0)  ←登録された。
centos/7          (virtualbox, 1804.02)
```

以上でKaliのBoxの作成が完了しました。

●Kaliの仮想マシンをVagrantで実行する

KaliのBoxの動作を確認します。プロジェクトフォルダーを新たに作成し、カレントディレクトリをそこに移動します。

```
ipusiron@Garoa MINGW64 ~/.vagrant.d/boxes/Kali-Linux-2018.2
$ cd /c/vagrant

ipusiron@Garoa MINGW64 /c/vagrant
$ mkdir kali

ipusiron@Garoa MINGW64 /c/vagrant
$ cd kali
```

vagrant initコマンドでVagrantfileファイルを作成します。

```
ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant init Kali-Linux-2018.2
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.

ipusiron@Garoa MINGW64 /c/vagrant/kali
$ ll
total 4
-rw-r--r-- 1 ipusiron 197121 3094 7月   7 18:32 Vagrantfile ← Vagrantfileファイルがある。

ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant up ← 仮想マシンを起動する。
（略）
ipusiron@Garoa MINGW64 /c/vagrant/kali
```

```
$ vagrant status
Current machine states:

default                    running (virtualbox)
（略）
```

vagrant upコマンド時にエラーが発生するときは、仮想マシンの設定とVagrantfileファイルの内容が食い違っている可能性があります。

例えば、vagrant up実行時に "default: SSH auth method: private key" で止まり、タイムアウトが発生するときは、SSHでアクセスできていません。しかし、vagrant statusコマンドで確認するとrunningになっているので、仮想マシンはすでに起動しています。VirtualBoxからVagrantの仮想マシンにログインして、修正してから再びvagrant upコマンドを試します。

もし共有フォルダーやVirtualbox Guest Additionsなどのエラーが出ていても、先に進めます。

vagrant upコマンドに成功したら、仮想マシンにアクセスします。

```
ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant ssh  ←仮想マシンにアクセスする。
Linux kali 4.16.0-kali2-amd64 #1 SMP Debian 4.16.16-2kali1 ↵
(2018-06-25) x86_64

The programs included with the Kali GNU/Linux system are free ↵
software;
the exact distribution terms for each program are described in ↵
the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
vagrant@kali:~$ id
uid=1001(vagrant) gid=1002(vagrant) groups=1002(vagrant),27(sudo)
vagrant@kali:~$ exit
```

```
logout
Connection to 127.0.0.1 closed.

ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant halt  ←仮想マシンを終了する(電源オフ)。
==> default: Attempting graceful shutdown of VM...
```

以上で、Boxの動作を確認できました。

Vagrantの共有フォルダー

Vagrantはホスト OSとゲスト OS間の共有フォルダーをサポートしています。デフォルトではプロジェクトフォルダーが共有されています。

ここでは、作成済みのKaliの仮想マシンを使って、共有フォルダーの動作について確認してみます。

①共有フォルダーを有効にする

Vagrantfileファイルにて、共有フォルダーの設定項目である「config.vm.synced_folder」のコメントアウトを外します。この項目の書式は次の通りです。

```
config.vm.synced_folder "<ホストOS側のパス>", "<ゲストOS側のパス>"
```

ホストOS側の共有フォルダーのパスは、Vagrantfileファイルからの相対パスで指定します。例えば、「ホストOS側にてVagrantfileファイルがあるフォルダー」を共有フォルダーにして、「ゲストOS側の"/vagrant"」に対応付けるには、次のように指定します。

```
config.vm.synced_folder "./", "/vagrant"
```

②仮想マシンを起動する

以上で共有フォルダーを有効にできたはずなので、仮想マシンを起動します。ここではKaliの仮想マシンを用います。

```
ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant up
（略）
$ vagrant ssh
（略）
vagrant@kali:~$ ls /                              "/vagrant"ディレクトリが見える。
0     etc              lib         media  proc  srv  vagrant  ←
bin   home             lib32       mnt    root  sys  var
boot  initrd.img       lib64       opt    run   tmp  vmlinuz
dev   initrd.img.old   lost+found  out    sbin  usr  vmlinuz.old
```

③共有したファイルが参照できることを確認する

ホストOS側で配置したファイルが、ゲストOSで参照できることを確認します。"C:¥vagrant¥kali" フォルダーに "test.txt" ファイルを配置します。ここでは、次のような内容にしました。

"test.txt"ファイル

```
Hi.
共有フォルダーに配置したファイルです。
```

仮想マシン側でtest.txtファイルの存在を確認できました。

```
vagrant@kali:~$ cd /vagrant
vagrant@kali:/vagrant$ ls
test.txt  Vagrantfile
```

catコマンドを使って内容が表示されることを確認します（図8）。

図8　Vagrantで共有されたファイル

Shift-JISで保存すると日本語が文字化けします。一方、UTF-8で保存すると、文字化けしません。

● 共有フォルダー内のパーミッション

Vagrantではゲスト OS側からファイルやフォルダーのパーミッションが変更できないようになっています。「config.synced_folder」項目において、dmodeでディレクトリのパーミッション、fmodeでファイルのパーミッションを設定できます。

```
config.vm.synced_folder "./", "/vagrant", mount_options:
['dmode=777','fmode=755']
```

Vagrantfileファイルを変更した際には、vagrant reloadコマンドで仮想マシン

を再起動します。これにより、Vagrantfileファイルが再読み込みされます。

```
vagrant@kali:/vagrant$ ls -la /vagrant
total 8
drwxrwxrwx  1 vagrant vagrant    0 Jul 10 16:35 .
drwxr-xr-x 26 root    root    4096 Jul  7 18:57 ..
-rwxr-xr-x  1 vagrant vagrant   61 Jul 10 16:38 test.txt
drwxrwxrwx  1 vagrant vagrant    0 Jul  7 18:33 .vagrant
-rwxr-xr-x  1 vagrant vagrant 3180 Jul 10 17:01 Vagrantfile
```

ディレクトリのパーミッションは777。

ファイルのパーミッションは755。

●外部フォルダーを共有フォルダーにする

「config.synced_folder」項目を次のように設定すると、プロジェクトフォルダー（ここでは"kali" フォルダー）から見て外部に位置するフォルダー（"data" フォルダー）が共有フォルダーになります。

```
config.vm.synced_folder "../data", "/vagrant"
```

こうすることで、他の仮想マシンでも同じ共有フォルダーを自然に扱えます。また、プロジェクトフォルダーを削除しても、共有フォルダーは消えません。

⋙ Vagrantのネットワーク設定

Vagrantは、デフォルトでNAT接続の状態になっています（図9）。Vagrantで登録した仮想マシンは、VirtualBoxのメイン画面に表示されます。

図9　Vagrantで作成したKaliのネットワーク設定

NAT接続なので、仮想マシンからLAN内の端末（ホストも含む）、インターネットのサーバーにアクセスできます。しかし、別の仮想マシンへのアクセス、LAN内の端末から仮想マシンへのアクセスはできません。

Vagrantfileファイルの「config.vm.network」項目を設定することで、仮想LANアダプターが追加されます。さらに、仮想LANアダプターには、表1の3種類のネットワークオプションのいずれかを指定できます。

表1　Vagrantのネットワークオプション

ネットワークオプション名	指定文字列	VirtualBoxにおけるネットワークの種類
プライベートネットワーク	private_network	内部ネットワーク
パブリックネットワーク	public_network	ブリッジアダプター
ポートフォワーディング	forwarded_port	なし

VirtualBoxにおけるネットワークの種類を対応させるとわかりやすいといえます。

●プライベートネットワーク

- 静的IPアドレスを使う。
- IPアドレスを直接使ってアクセスする。
- 外部のPCは仮想マシンにアクセスできないので安全といえる。

- （同じ内部ネットワーク名に属する）仮想マシンはホスト自身と通信できる。

例："mynetwork" という内部ネットワーク名でプライベートネットワークに属する。
`config.vm.network "private_network", ip: "10.1.1.5", ↵` `virtualbox__intnet: "mynetwork"`

例：内部ネットワーク名を指定せずに、"true" を指定すると、intnet という名前が使われる。
`config.vm.network "private_network", ip: "10.1.1.5", ↵` `virtualbox__intnet: true`

● パブリックネットワーク

- 既存のLANにぶら下がる形で接続する。
- LAN内の端末とも通信できる。

例：静的IPアドレスを割り当てる。
`config.vm.network "public_network", ip: "192.168.1.10"`

例：DHCPで動的IPアドレスを割り当てる。
`config.vm.network "public_network"`

例：ブリッジするインターフェースを指定する。
`config.vm.network "public_network", bridge: '<インターフェース名>'`

　ブリッジするインターフェースを指定しないと、vagrant upコマンドの実行時に毎回確認されます。このときは数字を指定しますが、インターフェース名はその隣の文字列です。

例えば、「1) Intel(R) Centrino(R) Advanced-N 6205」と表示されていたら、インターフェース名は「Intel(R) Centrino(R) Advanced-N 6205」になるので、次のように記述します。

```
config.vm.network "public_network", bridge: 'Intel(R) Centrino(R) ↵
Advanced-N 6205'
```

● ポートフォワーディング

- ホスト側のポートへのアクセスを、ゲスト側のポートへのアクセスに変換する。
- IPアドレスなしでゲストへのアクセスを許可できる。
- ホストでは1024以下のポートをフォワードできない。

● ネットワークオプションを混在させてみる

1つの仮想マシンにて、これらのネットワークオプションを混在できます。ここではプライベートネットワーク、パブリックネットワーク、ポートフォワーディングのすべてを設定してみます。

①ネットワークオプションを指定する

Vagrantfileファイル内で、次のようにネットワークオプションを指定します。

```
config.vm.network "forwarded_port", guest: 80, host: 8083, host_
ip: "127.0.0.1"
config.vm.network "private_network", ip: "192.168.100.10"
config.vm.network "public_network"
```

②仮想マシンに設定を反映させる

仮想マシンを再起動して、設定を反映させます。

```
ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant reload
```

```
(パブリックネットワークでのアダプターを聞かれるので、ここでは1を入力する)
ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant ssh
```

ifconfigコマンドで確認すると、eth0〜eth2の3つのインターフェースが表示されます。eth0はNAT（デフォルト状態)、eth1はプライベートネットワークの静的IPアドレス、eth2はパブリックネットワークの動的IPアドレスになります（図10)。

```
vagrant@kali:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fec5:d1c  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:c5:0d:1c  txqueuelen 1000  (Ethernet)
        RX packets 351  bytes 38235 (37.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 277  bytes 39409 (38.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.100.10  netmask 255.255.255.0  broadcast 192.168.100.255
        inet6 fe80::a00:27ff:fe60:72e7  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:60:72:e7  txqueuelen 1000  (Ethernet)
        RX packets 5  bytes 1348 (1.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 24  bytes 1856 (1.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 16  base 0xd240

eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.11  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 240d:0:2b07:cc00:a00:27ff:fe22:7873  prefixlen 64  scopeid 0x0<glo
bal>
        inet6 fe80::a00:27ff:fe22:7873  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:22:78:73  txqueuelen 1000  (Ethernet)
        RX packets 139  bytes 20981 (20.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 49  bytes 5241 (5.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 17  base 0xd260

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 18  bytes 786 (786.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 786 (786.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

vagrant@kali:~$
```

図10　ifconfigコマンドの出力結果

Pingで疎通確認します。

```
vagrant@kali:~$ ping 192.168.1.1
vagrant@kali:~$ ping akademeia.info
```

③ポートフォワーディングが有効になっていることを確認する

ポートフォワーディングが有効になっていることを確認するために、Kali側でApacheを起動します。

```
vagrant@kali:~$ sudo service apache2 start
```

ホストOS側でブラウザを起動して、URL欄にhttp://127.0.0.1:8083を入力します。すると、Apacheの画面が表示されます（図11）。これはホストOSへのポート8083へのアクセスが、ゲストOSへのポート80にフォワードされたためです。
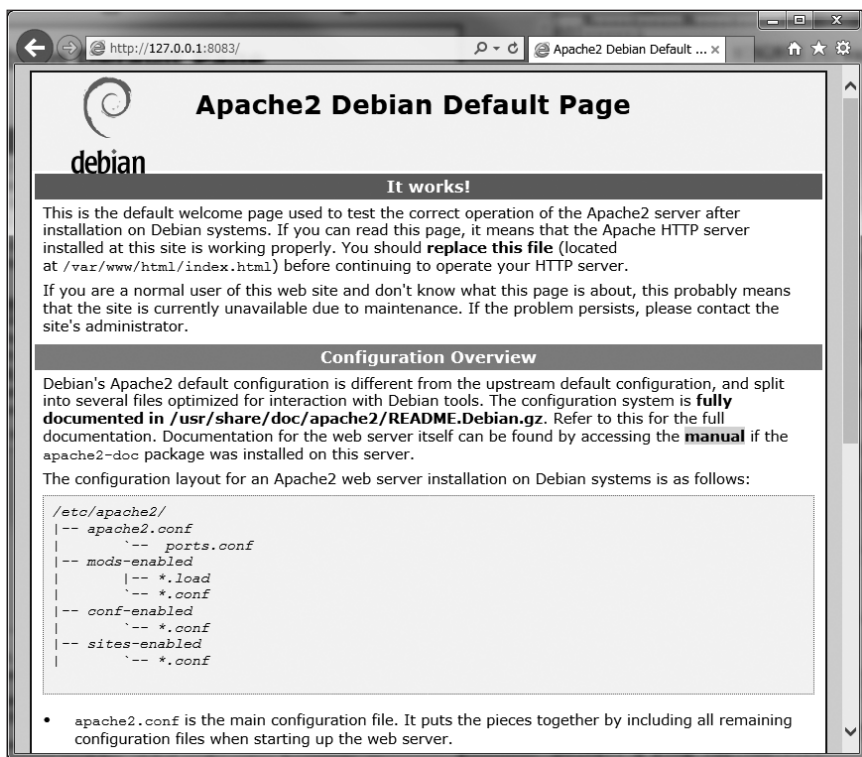
図11　ポートフォワーディングされた結果

Boxに対するコマンド

これまでは一連の手順の中でBoxに対するコマンドを説明してきました。ここでは、その復習とまだ説明していないコマンドについて紹介します（表2）。

表2　Boxに対するコマンド

コマンド	実行結果
vagrant box add <Box名>	Boxを追加する。
vagrant box list	ダウンロード済みのBoxを列挙する。
vagrant box remove <Box名>	不要なBoxを削除する。
vagrant box update	Boxをアップデートする。
vagrant box repackage <Box名> <プロバイダー> <バージョン>	再パッケージ化する。 Addした時点での状態を"package.box"として出力する。

Vagrantプラグインを追加する

Vagrantにプラグインを追加することで、機能を拡張できます。インストールされているプラグインを確認するには、次のコマンドを用います。

```
$ vagrant plugin list
```

ここでは便利なプラグインの一部を紹介します。

●vagrant-vbguest

dotless-de/vagrant-vbguest
https://github.com/dotless-de/vagrant-vbguest

仮想マシンにVirtualBox Guest Additionsがインストールされているかを確認し、必要があれば自動的にインストールしてくれるプラグインです。VirtualBoxのバージョンとBoxにインストールされているVirtualBox Guest Additionsのバージョンが違う場合にも最新化されます。結果として、手動でインストールする手間から解放されます。

次のコマンドでインストールできます。

```
$ vagrant plugin install vagrant-vbguest
```

●vagrant-hostmanager

devopsgroup-io/vagrant-hostmanager
https://github.com/devopsgroup-io/vagrant-hostmanager

ホストOSからゲストOSにホスト名でアクセスするためには、ホストOSのhostsファイルに、ゲストOSのホスト名とIPアドレスが登録されていなければなりません。これを手動で書き加えるのは手間です。

vagrant-hostmanagerプラグインを導入すれば、vagrant upコマンドを実行するとホストOSのhostsファイルに情報を登録してくれます。逆に、vagrant haltコマンドを実行すると、hostsファイルから情報を削除してくれます。

次のコマンドでインストールできます。

```
$ vagrant plugin install vagrant-hostmanager
```

コラム Vagrantでスナップショット

Vagrantはバージョン1.8から、スナップショット機能が追加されました。スナップショット機能とは、ある時点の仮想マシンの状態を保存したり復元したりする機能です。

これまではスナップショット機能を実現するために次のプラグインがよく使われていました。今はVagrant本体だけでスナップショット機能を実現できます(*9)。

*9：使い慣れたプラグインを使い続けるという選択肢もあります。

vagrant-vbox-snapshot
https://github.com/dergachev/vagrant-vbox-snapshot

sahara
https://github.com/jedi4ever/sahara

例えば、プロビジョニング（次項参照）用のスクリプトを作成したいとします。最初にスナップショットを作っておいてから、コマンドを追加して、vagrant provisionコマンドを実行します。その後、別のコマンドを追加して、vagrant provisionコマンドを実行します。これを繰り返して、1行ずつスクリプトを作っていきます。最後に通しでプロビジョニングが成功するかをチェックするために、スナップショットを復元してvagrant provisionコマンドを実行します。

プロビジョニングで自動化する

プロビジョニング（provisioning）とは、仮想マシンの状態の設定を自動化する機能です。最も簡単かつ基本的な方法は、シェルスクリプトでプロビジョニングを実現する方法です。シェルスクリプトのプログラミングスキルも流用できます。

①仮想マシンを停止する

仮想マシンが動作しているのであれば、停止します。

```
ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant halt
```

②プロビジョニングを有効にする

プロビジョニングを有効にするために、Vagrantfileファイル内を次のように変更します。

"Vagrantfile"ファイル（編集前）

```
# config.vm.provision "shell", inline: <<-SHELL
```

"Vagrantfile"ファイル（編集後）

```
config.vm.provision "shell", path: "provision.sh"
```

③シェルスクリプトを配置する

Kaliのプロジェクトフォルダーに次のシェルスクリプトを配置します。ここでは例として、文字列のアスキーアートを表示するfigletコマンドをインストールしてみます。

"provision.sh"ファイル

```
#!/bin/sh

echo "Installing figlet."
sudo apt install figlet --yes
```

④プロビジョニングを実行する

仮想マシンを起動してから、vagrant provisionコマンドでプロビジョニングを実行します。

```
ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant up
（略）
ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant provision
（略）
```

インストールが始まれば、"Installing figlet." のメッセージの後に通信や依存

パッケージのログが出力されるはずです（図12）。

```
ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant provision
==> default: Running provisioner: shell...
    default: Running: C:/Users/ipusiron/AppData/Local/Temp/vagrant-shell20180710
-42464-uibt12.sh
==> default: Installing figlet.
==> default: WARNING:
==> default: apt
==> default:
==> default: does not have a stable CLI interface.
==> default: Use with caution in scripts.
==> default: Reading package lists...
==> default: Building dependency tree...
==> default: Reading state information...
==> default: The following packages were automatically installed and are no long
er required:
==> default:   dh-python geoip-database-extra libbabeltrace-ctf1 libcamel-1.2-60
 libcue1
==> default:   libedataserver-1.2-22 libedataserverui-1.2-1 libfile-copy-recursi
ve-perl
==> default:   libhttp-parser2.7.1 libisl15 libjs-openlayers libllvm5.0 libnfs8
libsynctex1
==> default:   libtcl8.5 libtk8.5 libx265-146 libx265-151 openjdk-9-jdk
==> default:   openjdk-9-jdk-headless python-unicodecsv python3-configargparse
==> default:   python3-editorconfig python3-flask python3-itsdangerous python3-j
sbeautifier
==> default:   python3-pyinotify python3-simplejson python3-werkzeug tk8.5
==> default: Use 'sudo apt autoremove' to remove them.
==> default: The following NEW packages will be installed:
==> default:   figlet
==> default: 0 upgraded, 1 newly installed, 0 to remove and 117 not upgraded.
==> default: Need to get 136 kB of archives.
==> default: After this operation, 756 kB of additional disk space will be used.
==> default: Get:1 http://ftp.ne.jp/Linux/packages/kali/kali kali-rolling/main a
md64 figlet amd64 2.2.5-3 [136 kB]
==> default: dpkg-preconfigure: unable to re-open stdin: No such file or directo
ry
==> default: Fetched 136 kB in 2s (55.7 kB/s)
==> default: Selecting previously unselected package figlet.
==> default: (Reading database ...
(Reading database ... 20%abase ... 5%
(Reading database ... 45%abase ... 25%
==> default: (Reading database ... 50%
==> default: (Reading database ... 55%
==> default: (Reading database ... 60%
==> default: (Reading database ... 65%
==> default: (Reading database ... 70%
==> default: (Reading database ... 75%
==> default: (Reading database ... 80%
==> default: (Reading database ... 85%
==> default: (Reading database ... 90%
==> default: (Reading database ... 95%
(Reading database ... 407579 files and directories currently installed.)
==> default: Preparing to unpack .../figlet_2.2.5-3_amd64.deb ...
==> default: Unpacking figlet (2.2.5-3) ...
==> default: Setting up figlet (2.2.5-3) ...
==> default: update-alternatives: using /usr/bin/figlet-figlet to provide /usr/b
in/figlet (figlet) in auto mode
==> default: Processing triggers for man-db (2.8.3-2) ...

ipusiron@Garoa MINGW64 /c/vagrant/kali
$
```

図12　vagrant provisionコマンドの実行結果

⑤インストールできたことを確認する

figletがインストールできたことを確認します（図13）。

```
ipusiron@Garoa MINGW64 /c/vagrant/kali
$ vagrant up
（略）
vagrant@kali:~$ figlet Happy hacking.
```

図13　figletコマンドの実行

ここではソフトウェアのインストールを自動化しましたが、他にも色々できます。同じ処理はどんどん自動化させることをおすすめします。操作ミスもなくなり、効率も上がります。

コラム　puPHPet.comを利用したプロビジョニング

Webアプリの開発用にVagrantを使うのであれば、puPHPet.comが便利です。Web上で設定した内容のBoxとプロビジョニング用スクリプトを生成してくれます。

PuPHPet
https://puphpet.com/

Vagrantのアップデート

Vagrantのインストーラーを起動します。旧バージョンと同じフォルダーを指定してインストールします。インストールが完了したら、PCを再起動します。後は、vagrant -vコマンドでバージョンを表示して、アップデートの成否を確認します。